Name: ______________________________ Andrew Id: ____________________

15-112 Spring 2023 Quiz 10 (Optional)

Up to 20 + 10 minutes (finish within 20 minutes for 20% proficiency bonus). No calculators, no notes, no books, no computers. Show your work!

Complexity of Python Built-ins

General	
Function/Method	**Complexity**
Print	O(N)
Range in Iteration	Number of iterations = (end - start)/step

Strings: s is a string with N characters	
Function/Method	**Complexity**
Len	O(1)
Membership (`in`)	O(N)
Get single character	O(1)
Get slice	O(end - start)
Get slice with step	O((end - start)/step)
`chr()` and `ord()`	O(1)
Concatentation	O(len(s1) + len(s2))
`upper()`, `lower()`	O(N)
`isalpha()`, `isalnum()`, `isdigit()`	O(N)
String Edit Methods	O(N)
Substring Search Methods (`in`)	O(N)
Count	O(N)

Lists: L is a list with N elements	
Function/Method	**Complexity**
Len	O(1)
Append	O(1)
Extend	O(K)
Concatentation with +=	O(K)
Concatentation with +	O(N + K)
Membership Check	O(N)
Pop Last Value	O(1)
Pop Intermediate Value	O(N)
Count values in list	O(N)
Insert	O(N)
Get value	O(1)
Set value	O(1)
Remove	O(N)
Get slice	O(end - start)
Get slice with step	O((end - start)/step)
Sort	O(N log (N))
Multiply	O(N*D)
Minimum	O(N)
Maximum	O(N)
Copy	O(N)
Deep Copy	O(N*M)

Sets: s is a set with N elements	
Function/Method	**Complexity**
Len	O(1)
Membership	O(1)
Adding an Element	O(1)
Removing an Element	O(1)
Union	O(len(s) + len(t))
Intersection	O(min(len(s), len(t)))
Difference	O(len(s))
Clear	O(len(s))
Copy	O(len(s))

Dictionaries: d is a dictionary with N key-value pairs	
Function/Method	**Complexity**
Len	O(1)
Membership	O(1)
Get Item	O(1)
Set Item	O(1)
Delete Item	O(1)
Clear	O(N)
Copy	O(N)

1. (12 points) **Big-O:** For each function shown below, write the total Big-O runtime of the function in terms of N, the length of the function argument, in the box to the right of the code. All answers must be simplified- do not include lower-order terms!

```python
def f1(s):
    countAB = s.count('a') * s.count('b')
    return countAB
```

```python
def f2(s):
    total = 0
    for c in s:
        total += s.count(c)
    return total
```

```python
def f3(s):
    count = 0
    for c in 'abcdefghijklmnopqrstuvwxyz':
        count += s.count(c)
    return count
```

```python
def f4(L):
    n = len(L)
    count = 0
    for i in range(n):
        if i not in L:
            for j in range(n):
                if j*i not in L:
                    count +=1
    return count
```

```python
def f5(L):
    M=[]
    for i in range(len(L)):
        for j in range(i, len(L)):
            M.append(L[i]*L[j])
    M.sort()
    return M
```

```python
def f6(L):
    S = set(L*len(L))
    for e in S:
        if e**2 in S:
            return True
    return False
```

```python
def f7():
    n = len(L)
    result = 0
    for i in range(2*n, n**2, 3):
        if (i not in L):
            for j in range(i, n):
                result += (i * j)//n
    return result
```

```python
def f8(L):
    n = len(L)-1
    while(n > 0):
        if L[n] % 2 == 0:
            return True
        n = n // 2
    return False
```

2. (8 points) **Free Response: Find Zero Triplets**

   Write the function `findZeroTriplets(L)` that takes as input a list `L` of integers of length `N` and returns a set of all triplets in the list whose sum is equal to 0. For example, if the given list is `[-1, 0, -3, 2, 1]`, you should return `{(-1, 0, 1), (-3, 1, 2)}`. Note that triplets must be sorted and that you can't use the same element multiple times unless there are enough copies in the list. If there is no valid triplet, you should return the empty set. You may assume that `L` is a list containing only integers. The *naive* solution would use three loops to check all triplets of values in `L`, but you must do better than that for full credit.

   Here are more test cases:

```
assert(findZeroTriplets([0, 0]) == set())  # not enough elements
assert(findZeroTriplets([0, 0, 0]) == {(0,0,0)})  # ok
assert(findZeroTriplets([0, 0, 1]) == set())  # no valid triples
assert(findZeroTriplets([-1, 0, -3, 2, 1]) == {(-1, 0, 1), (-3, 1, 2)})
assert(findZeroTriplets([-1,2,0]) == set())  # there are no valid triplets
assert(findZeroTriplets([-1, 2, 0, -1]) == {(-1,-1, 2)})  # there's only one valid triplet
```

   You MUST use sets and dictionaries to do this faster than $O(N^3)$